Thumbnail Streaming

R.Hauser, A. Yurkewicz
Version 1.3

November 27, 2002

1 Introduction

This note describes how to use the `np_tmb_stream` package to implement streaming on thumbnail files. Running the executable in this package over a data set will produce multiple output files where events can be selected by trigger bits, particle types and their properties, and logical combinations of them.

2 Use Cases

The package has been developed for the use in the NP physics group, although it should be useful for other groups and general users as well.

The techniques described will be useful in any analysis where a user writes out a subset of events based on some arbitrary criteria.

The following scenarios were considered:

- A physics group decides which streams to produce. The program is run on a regular basis (e.g. daily) on the latest reconstructed thumbnail data sets and the output is made available to everybody in a central place.
- A user decides she wants to add or tighten some cuts and produce yet another private version of the thumbnail files for private use. She runs e.g. over a single stream from the physics group and writes the output to a local scratch disk.
- A user wants her own specialized selection of events which cannot be obtained with the existing executable. It should be easy to extend the program to accomodate any desired selection cuts.
- Except for the last case, all parameters should be easily customizable via RCP so that no recompilation is necessary.
- It should be possible to make a single run over the input and produce all desired output files in one step.

3 Tags and WriteEvent

All of the above can be achieved with a combination of features available in the D0 framework. We make use of two of them:

- the capability to *tag* an event with an arbitrary string
- the capability to have more than one instance of the `WriteEvent` package and to customize each instance to write only events which have certain tags

Tags are essentially strings which can be attached to an `edm::Event` as it flows through the framework. Tags can be attached to a constant event, so they can be used e.g. in an analyze package which isn't supposed to modify an event.

Tagging an event is trivial:

```
fwk::Result MyPackage::analyzeEvent(const edm::Event& event)
{
    ...
    if(some_condition) {
       event.tag(edm::Tag("MyTag"));
    }
    ...
}
```

The `io_package/WriteEvent` package is usually used to write out events in the EVPACK format. The `WriteEvent` package can be customized via its RCP file to output only events which have certain tags:

```
string PackageName = "WriteEvent"

string OutputFile = "myEvents"

// ...things omitted

bool UseOutputFilter = true

string WriteEventTags = "MyTag"
```

The example above will only write the tagged events into an output file called *myEvents*. Multiple tags can be given in the `WriteEventTags` parameter.

The streaming program uses this technique by implementing a set of simple packages which will tag events based on trigger bits, particle types and some of their properties. It also allows combining multiple tags into a new output tag.

4 Tags based on Particle Types

There are four framework packages available to tag events based on particle types:

- EM particles
- Jets
- Muons
- Missing ET

For each type there is a restricted set of cuts that can be parameterized in the corresponding RCP file. These parameters were chosen by

- Looking at the top group streaming example.
- Going through about a dozen presentations in the NP group and looking at what selection cuts people had used.

For particles, the multiplicity is one parameter. Note that each of the tagging packages can be instantiated more than once with a different set of parameter values. So if one wants a single 30 GeV jet and a 2 x 20 GeV dijet, one simple creates two different RCP files and instantiates the `JetTag` package twice in the main RCP file.

The `NumParticles` parameter is required. The other parameters can be either absent or a vector of values, one for each particle. For backward compatibility a single parameter is interpreted as a vector of size one.

4.1 EM Particles

The `EMTag` package implements tagging for EM particles. The following selections can be applied:

int NumParticles The number of particles required.

float Thresholds A vector of pT thresholds, one for each particle.

float EMfraction

float Isolation

float HMx8

Furthermore the algorithm type (scone, cellNN) can be specified.

The tag which is assigned to the event is specified in

```
string Tag = "MyTag"
```

This is the same for all the following packages.

4.2 Jets

The `JetTag` package implements tagging for jets. The following selections can be applied:

int NumParticles Number of jets required

float Thresholds vector of pT threshold for each jet

float EMfraction

float CHfraction

float HOTfraction

int N90

Furthermore the algorithm type and its parameters can be specified.

4.3 Muons

The `MuonTag` package implements tagging for muons. The following selections can be applied:

int NumParticles Number of jets required

float Thresholds vector of pT threshold for each jet

int Nseg

4.4 Missing ET

The `METTag` package implements tagging for missing ET. Only a single threshold can be specified:

```
float Threshold = ...
```

5 Tags based on Trigger Bits

The default executable includes the `D0TriggerFilter` from the `analysis␣utitilies` package. It can be used to filter out events which have passed specified triggers. This speeds up processing since there is no thumbnail unpacking done if the event doesn't pass this first filter.

The `D0TriggerSelector` classes are also used to assign tags to events with certain trigger bits.

The `TriggerTag` package implements the tagging of events depending on trigger bits. The only parameter is the list of trigger names and the output tag:

```
string PackageName = "TriggerTag"

// the tag to produce for the event
string Tag = "EMTRIG"

string TriggerNames = ( "EM_HI" "EM_LO" )
```

6 Combining Tags

The `WriteEvent` package will write out events based on a set of tags. This provides a logical OR for defining a stream. One also needs a logical AND to combine the results of the simple packages described above.

The `AndTag` package combines existing tags into new ones. It requires that all input tags are present on an event and then tags the event with a new tag. The only parameters are the output tag and the list of input tags:

```
string PackageName = "AndTag"

string Tag = "E+J"

string Input = ( "1EM" "1JET" )
```

7 Tags based on Cut Expressions

This is a new way of tagging events based on a list of cuts for different particle types. It reduces the number of RCP files you need and is more flexible in the selections it allows. In many cases it should be possible to have a single RCP file for a stream where before multiple files plus a `AndTag` RCP was needed.

It is based on the `cuts` package which is part of the D0 release. It makes attributes of the various objects in an event easily available and allows the user to specify an arbitrary expression in terms of these attributes. The following example shows the basic structure of the RCP file:

```
string PackageName = "ObjectTag"

// the tag to produce for the event
string Tag = "EM+JET"

// list of trigger names, empty means no trigger selection applied
string Trigger = ( "EM_HI" )
```

```
// The list of cuts to apply
string Cuts = ( "Cut1" "Cut2" )

string Cut1 = ( "EM"  "Pt > 15.0 && emfrac > 0.95")
string Cut2 = ( "JET" "Pt > 20.0 && emETfraction < 0.9 && HM8 < 20" )

// simple jet name
string JetName = "JCCB"
```

As can be seen, it combines various features which are only available in separate pieces when using the old method.

- A trigger selection cut can be applied for this set of cuts. This is the first criteria to pass.
- A list of selection cuts can be given.
- The cuts can be applied to different particle types.
- The number of attributes available is much larger (and easier to extend).
- The user can define an arbitrary expression using these attributes.

The event is tagged, if the trigger condition and all cuts are satisfied.

A cut consists of two strings: the object type and a boolean expression. The object type must be one of `EM, MU, JET, MET, CHP`. The available attributes depend on the type of object. They are usually named after the corresponding method of the C++ class or the quality object associated with it.

The following attributes are available:

All particles Px, Py, Pz, E, Pt, Phi, Eta, AbsEta, Theta, P, Charge, TypeId

EM emfrac, isolation, coreE, isoE, nb_ps, nb_tracks, ps_matching

MU nseg, ncentralmatch, centralrank, whits_A, whits_bc, shits_a, shits_bc, chisq, chisqloc, qptloc, qualityloc, e33, e55, EInCone1, EInCone2, EInCone3, EInCone4, Loose, Medium, Tight

JET emETfraction, chETfraction, hotcellratio, n90, HM8

MET MET, SET, MEx, MEy, MET_noeta, SET_noeta, MEx_noeta, MEy_noeta, MET_weta, SET_weta, MEx_weta, MEy_weta

8 Putting It All Together

8.1 Customizing the RCP Files

The RCP files are divided in three parts, one each for input, tagging and output.

Here is the main RCP file `runTMBStream.rcp` which you normally shouldn't have to modify:

```
string InterfaceName = "process"
string Packages = "input tag output"

RCP input  <np_tmb_stream read>
RCP tag    <np_tmb_stream tags>
RCP output <np_tmb_stream write>
```

Here is the input part, where you can customize the trigger selector.

```
string Packages = "geo sam read config trigsel unptmb links"

RCP geo        = <geometry_management geometry_management>
RCP read       = <d0reco D0recoReadEvent>
RCP sam        = <np_tmb_stream TMBSAMManager>
RCP config     = <run_config_fwk RunConfigPkg>
RCP trigsel    = <np_tmb_stream D0TriggerFilter>
RCP unptmb     = <thumbnail UnpThumbNail>
RCP links      = <linked_physobj  LinkedPhysObjReco>
```

The only thing you should have to customize here is the trigger selection. The corresponding file is `D0TriggerFilter` to enable/disable it and specify a list of trigger names. Note that usage of this package is not required.

The `tags.rcp` file simply consists of a list of packages where you specify your rcp files with the appropriate cuts. The same for the `write.rcp` file and output specifications rsp.

Let's assume you want the following streams (taken from the top examples):

- 5 GeV muon and 10 GeV jet
- two 2 GeV muons
- 18 GeV em object and 10 GeV jet

For each distinct particle you should create an RCP file which defines the cuts for it. In the example above this would be:

- one 5 GeV muon
- two 2 GeV muons

- one 10 GeV jet
- one 18 GeV em

Modify the existing examples for each particle type and produce the following RCP files (with the chosen output tags shown as well):

- `muon1-5.rcp`, tag is `MU1`
- `muon2-2.rcp`, tag is `MU2`
- `jet1-10.rcp`, tag is `JET`
- `em1-18.rcp`, tag is `EM`

Now modify the `tags.rcp` file as follows:

```
string Packages = "muon1 muon2 jet em"
```

and add:

```
RCP muon1 = <np_tmb_stream muon1-5>
RCP muon2 = <np_tmb_stream muon2-2>
RCP je    = <np_tmb_stream jet1-10>
RCP em    = <np_tmb_stream em1-18>
```

Some of the tags already correspond directly to one of our output streams (the dimuon). For others we have to combine the existing tags. Note that the 10 GeV jet tag is used for two output streams.

We do this by adding two instances of `AndTag`:

```
string Packages = "muon1 muon2 jet em emjet mujet"

RCP emjet = <np_tmb_stream ejet>
RCP mujet = <np_tmb_stream mujet>
```

where e.g. the `ejet.rcp` file whould contain the following:

```
string PackageName = "AndTag"

string Tag = "E+J"
string Input = ( "EM" "JET" )
```

and similarly for the `mujet` RCP file.

All that is left now is to instantiate `WriteEvent` three times. We do this in the `write.rcp` file:

```
string Packages = "emjetWrite mujetWrite mu2Write"

RCP emjetWrite = <np_tmb_stream ejetWrite>
RCP mujetWrite = <np_tmb_stream mujetWrite>
RCP mu2Write   = <np_tmb_stream mu2Write>
```

Here is a snippet of what you would change in such a file from the default values:

```
string OutputFile = "emjet"

bool UseOutputFilter = true

string WriteEventTags = "E+J"
```

and similarly for the dimuon and muon plus jet files.

You can restrict the number of events which are written per output file. If you are running over large data sets, you might want to write the results in more than one file. The `mkstream.sh` script in the `rcp` directory will produce a proper configuration file for `WriteEvent`. You call it with a *stream name*, the maximum number of events per file, the *output filename* and a list of tags:

```
% ./mkstream.sh  MyStream 1000 myoutput tag1 tag2...
```

This will create a new RCP file and all you have to add is `MyStream` as a package in your main RCP file. The output will go into files called `myoutput-00, myoutput-01, ...`, where no file should have more than 1000 events.

From p13 on you can also use new features of the `WriteEvent` package. The output file name may contain special characters preceded by a % sign. E.g. the `%n` pattern expands into a sequential number for each new output file that is written. So

```
string OutputFile = "dimuon-%n"
```

achieves the same effect as the script above in a much simpler way. See the `io_packages/PatternExpander.hpp` file for more options.

8.2 Using SAM

To use SAM instead of a normal input file (list), change the variable `bool UseSAM = 1` in `rcp/TMBSAMManager.rcp` to `bool UseSAM = 0` and use a SAM project definition instead of a file name.

8.3 Running It

To use the program, check out the `np_tmb_stream` package first and modify the RCP files for your needs.

```
% setup n32                      # only on d0mino
% setup D0RunII t02.30.00        # choose a recent version...
% newrel -t t02.30.00 work
% cd work
% d0setwa
% addpkg -h np_tmb_stream

% gmake np_tmb_stream.all        # only if there is no default executable
                                  # it will also take almost 1 GB of memory
                                  # to link so be patient...
```

To run the program over a single thumbnail file, just do:

```
TMBStream_x -rcp np_tmb_stream/rcp/runTMBStream.rcp
            -input_file <your_file>
```

To run the program with SAM the most convenient way is to use `d0tools`:

```
% setup d0tools

# the following in one line...

% rund0exe -exe=TMBStream_x -rcp=runTMBStream.rcp       \
           -rcppkg=np_tmb_stream -localbuild -localrcp  \
           -defname=tmbfilestest -batch -num=200
```

where tmbfilestest is the name of your SAM project.

In most cases you will check out the package locally and modify some of its rcp parameter. The `scripts/runTMBstream.sh` script calls `rund0exe` with most parameters already setup.

9 Production Streaming

Once the thumbnail data set becomes very large, it is very cumbersome to either have a single batch jobs process all events (because of the time it takes) or to split the dataset artificially in multiple sets and keep track of which have been processed etc.

A dataset definition normally corresponds to a growing set of files. When skimming events, the process should only run over files it hasn't processed so far. The following procedure achieves this without much user intervention.

First define your SAM dataset as usual. This definition should contain the dimensions you are interested in. The file list in this set will typically have changed every time you start a new project and ask for the newest snapshot.

Here is an example for the definition:

```
((VERSION p11.09.%,p11.1% and DATA_TIER thumbnail) \
  and TRIG_CONFIG_TYPE physics) and  \
 TRIG_CONFIG_NAME %global%"
```

Let's say we call this definition `np_skim_v1`. Now we decide on the definitions of the event streams we want and their selections cuts and put all this in the rcp files. This set of parameters defines a certain version of our skimming process, let's say version 2. We now define a new SAM dataset which looks like this:

```
DATASET_DEF_NAME np_skim_v1 minus \
 ((PROJECT_NAME rh_proc_v2-% and \
   CONSUMED_STATUS consumed) and \
   CONSUMER rhauser)
```

Replace the project and user name with your own.

Let's call this definition `np_skim_v1_notprocessed`. All we have to do from now on is to use this new definition and a SAM project name which starts with `rh_proc_v2-`.

If you want to change your stream definitions, you should change your project name. If you change your original dataset definition (e.g. going from p11 to p13), you should change both dataset name and project name.

You can use `rund0exe` on the CAB cluster to have multiple processes consume files from the same project, thereby effectively parallelizing your job automatically:

```
setenv SAM_PROJECT rh_proc_v2-‘date +%Y%m%d-%H%M%S‘
rund0exe -cab -jobs=10 -defname=np_skim_v1_notprocessed ...
```

See the help section of `rund0exe` for other parameters that are needed with CAB.

10 Writing your own Package and Integrating it

If the selections provided by the examples above are not enough, the user can extend the program by simply adding another framework package of her own. As mentioned above, events can be tagged as part of a user's normal analysis.

The following is a recipe for a simple user-defined tagging package.

10.1 Header File

Create a new package and put in a header file like `MyExample.hpp`:

```
#ifndef MYEXAMPLE_HPP
#define MYEXAMPLE_HPP

#include "edm/Event.hpp"
#include "edm/TKey.hpp"
#include "edm/Tag.hpp"

#include "framework/Package.hpp"
#include "framework/hooks/JobSummary.hpp"
#include "framework/interfaces/Flow.hpp"

namespace TMBStream {

    /**
     * My tagging package
     */
    class MyTag : public fwk::Package,
                  public fwk::Tag,
                  public fwk::JobSummary {
    public:
        /// Package constructor
        MyTag(fwk::Context *context);

        /// Destructor
        ~MyTag();

        /// fwk::Tag interface
        virtual fwk::Result tagEvent(const edm::Event& event);

        /// fwk::JobSummary interface
        virtual fwk::Result jobSummary();

        /// Method for accessing the package name.
        std::string packageName() const {return package_name();}

        /// Method for accessing the package name.
        static const std::string package_name() {return "MyTag";}

        /// Method for accessing the version of the package.
        static const std::string version() {
           return "$Id: tmb_streaming.tex,v 1.4 2002/08/21
                    20:01:25 rhauser Exp $";
        }
```

```
    private:
        // output
        std::string _tag;           // tag to use for the event

        // selection criteria, more member variables if necessary

        // statistics
        int                _tagged_events; // number of tagged events
    };
}
#endif // MYEXAMPLE_HPP
```

Then create a source file to implement the various methods:

```
#include "myexample/MyExample.hpp"

#include "rcp/RCP.hpp"

#include "framework/Registry.hpp"

using namespace edm;
```

You should include any additional header files to access e.g. the chunks you are interested in.

```
namespace TMBStream {

    FWK_REGISTRY_IMPL(MyTag,"$Name:  $");

    // constructor
    MyTag::MyTag(fwk::Context *context)
        : fwk::Package(context),
          fwk::Tag(context),
          fwk::JobSummary(context),
          _tagged_events(0)
    {

        // access RCP database here
        edm::RCP rcp = packageRCP();
        _tag = rcp.getString("Tag");
    }
```

At this point you should initialize any other parameters from the RCP file.

The `tagEvent` method is the place which is called for each event and where you should decide to tag it or not:

```
    // fwk::Tag interface
    fwk::Result MyTag::tagEvent(const edm::Event& event)
    {
        if(some_condition_is_true) {
            event.tag(edm::Tag(_tag));
            _tagged_events++;
            return fwk::Result::success;       // no need to continue
        }
        return fwk::Result::success;
    }

    // fwk::JobSummary interface
    fwk::Result MyTag::jobSummary()
    {
        std::cout << "MyTag/" << _tag << " tagged events: "
                  << _tagged_events << std::endl;
        return fwk::Result::success;
    }
}
```

In addition you need a `RegMyTag.cpp` file:

```
#include "framework/Registry.hpp"

using namespace fwk;

namespace TMBStream {

FWK_REGISTRY_DECL(MyTag)

}
```

And finally you should add `RegMyTag` to the `src/OBJECT_COMPONENTS` and the `bin/OBJECTS` file.

All the examples of how to access physics objects from
`http://www-d0.fnal.gov/d0dist/dist/packages/analysis_example/devel/doc/`
apply to these examples, too.

11 Other Issues

11.1 Chunks

At the moment the example RCP files provided in `np_tmb_stream` write out only a selected set of chunks (given by the `WriteChunkList` parameter). These are:

```
string WriteChunkList = "MCKineChunk TMBTriggerChunk
                         thumbnail::ThumbNailChunk fwk::HistoryChunk"
```

This will leave out any additional chunks which were in the original input file, like the `CalDataChunk`.

11.2 Other Input Formats

The program as described is actually completely independent from the input format. It can be used e.g. for streaming DSTs as well, by simply changing the `unptmb` entry in the packages list and modifying the list of chunks to write.

11.3 Other Output Formats

The program could produce the ROOT tree files at the same time as it does the streaming on the thumbnail files. However, the current implementation of the root tree generation does not yet allow multiple output files or selection based on tags.